GSI3: Security for Grid Services

Von Welch[1](welch@mcs.anl.gov), Frank Siebenlist[2], Ian Foster[12], John Bresnahan[2], Karl Czajkowski[3], Jarek Gawor[2], Carl Kesselman[3], Sam Meder[1], Laura Pearlman[3], Steven Tuecke[2]

[1]University of Chicago, Department of Computer Science
[2]Argonne National Laboratory, Mathematics and Computer Science
[3]University of Southern California, Information Sciences Institute

Abstract

Grid computing is concerned with the sharing and coordinated use of diverse resources in distributed "virtual organizations." The dynamic and multi-institutional nature of these environments introduces challenging security concerns that demand new technical approaches. In particular, we must deal with diverse local mechanisms, support dynamic creation of services, and enable dynamic creation of trust domains. We describe how these issues are addressed in two generations of the Globus Toolkit (GT2). First, we review the GT2 approach; then, we describe in detail new approaches developed to support the GT3 implementation of the Open Grid Services Architecture, a new initiative aimed at recasting key Grid concepts within a service-oriented framework. GT3's security implementation uses WS-Security mechanisms for credential exchange and other purposes, and introduces a tight least privilege model that avoids the need for any privileged service.

1 Introduction

The term "Grid" refers to systems and applications that integrate and manage resources and services that are distributed across multiple control domains [12]. Initially pioneered in the e-science context, Grid technologies are also generating interest in industry, as a result of their apparent relevance to commercial distributed computing scenarios [14].

A common scenario within Grid computing is the formation of dynamic "virtual organizations" (VOs) [16] comprising groups of individuals and associated resources and services united by a common purpose but not located within a single administrative domain. The need to support the integration and management of resources within such VOs introduces challenging security issues [15]. For a variety of issues relating to certification, group membership, authorization, and the like, participants in such VOs represent an overlay with respect to whatever trust relationships exist between individual participants and their parent organizations, as well as with respect to whatever security mechanisms are in place at those parent organizations.

Grid computing research has produced security technologies based around not direct interorganizational trust relationships but rather the use of the VO as a bridge among the entities participating in a particular community or function. The results of this research have been incorporated into a software system called the Globus Toolkit®[1](GT) that is now seeing widespread use [4], and that uses public key technologies to address issues of single sign on, delegation [17], and identity mapping, while supporting standard APIs such as GSS-API [23]. The Grid Security Infrastructure (GSI) is the name given to the portion of the Globus Toolkit implementing its needed security functionality.

The recent definition of the Grid service specification and other elements of the Open Grid Services Architecture (OGSA) [14] within the Global Grid Forum (GGF) introduces new challenges and opportunities for Grid security. In particular, integration with Web services and hosting environment technologies introduces opportunities for integration with a variety of emerging security technologies, such as SAML and Web services security

Integration of GSI with OGSA enables the use of Web services techniques for policy publishing [2] allowing applications to automatically determine what security policies and mechanisms are required of them. Implementing security in the form of OGSA services allows them to be used as needed by applications to meet these requirements. Advanced hosting environments enable this functionality to be implemented outside of the application, simplifying development.

The remainder of this article is as follows. We review Grid security challenges in Section 2 and GT3 security mechanisms in Section 3. Then, in Sections 4 and 5, we introduce OGSA security mechanisms and our GT3 implementation. We conclude in Section 6 with a brief discussion of future work.

2 Grid Security Challenges

Security requirement within the Grid environment are driven by the need to support scalable, dynamic, distributed *Virtual Organizations* (VOs) [16]—collections of diverse and distributed individuals that seek to share and use diverse resources in a coordinated fashion resources. From a security perspective, a key attribute of VOs is that participants and resources are still members of their classical organizations and are governed by their rules and policies.

While some VOs, such as multiyear scientific collaborations, may be large and long-lived and will be created through negotiated access to resources with the resource providers, others may be short-lived, with little or no overhead desired. For example, two individuals may wish to share documents and data across their organizations as they write a proposal, in effect forming a VO to complete that single task. In such situations, the resources being coordinated will have no preexisting knowledge of each other but are coordinated only through their trust of the user.

A VO may be thought of as a *policy domain overlay*. Multiple resources or organizations outsource certain policy control(s) to a third party, the VO, which coordinates the outsourced policy in a consistent manner to allow for coordinated resource sharing and use.

Complicating Grid security is the fact that new services (i.e., resources) may be deployed and instantiated dynamically over a VO's lifetime. For example, a user may establish personal stateful interfaces to existing resources, or the VO itself may create directory services to keep track of VO participants. Like their static counterparts, these resources must be securely coordinated and interact with other services.

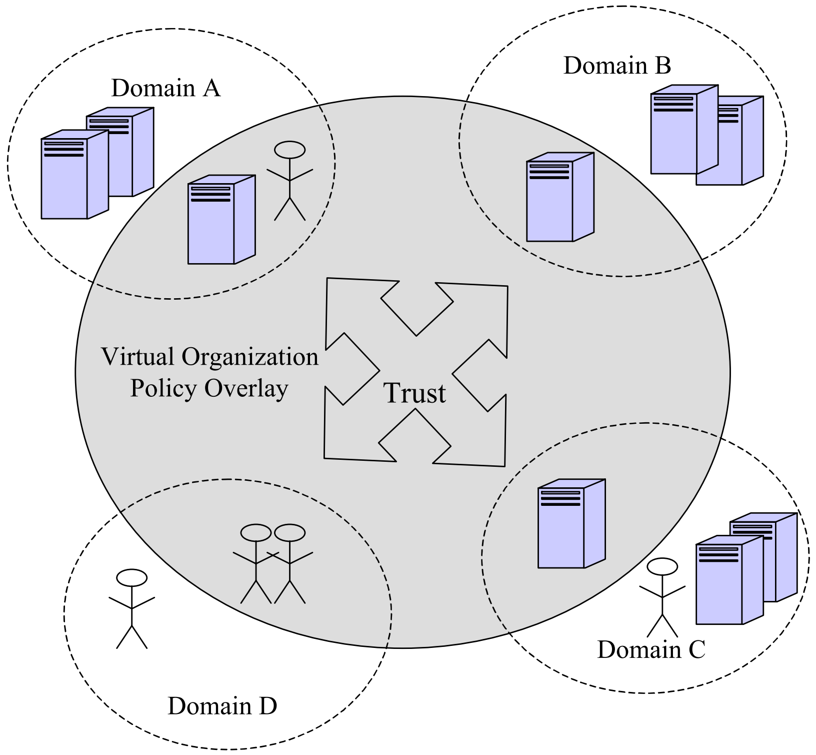

Figure 1: A Virtual Organization policy domain overlay pulls together participants from disparate domains into a common trust domain.

This combination of dynamic policy overlays and created entities drives the need for three key functions in a Grid security model.

1. *Multiple security mechanisms*. Organizations joining a VO often have significant investment in existing security mechanisms and infrastructure. Grid security must interoperate with, rather than replace, those mechanisms.
2. *Dynamic creation of services*. Users must be able to create new services (i.e., "resources") dynamically without administrator intervention. These services need to be coordinated and must interact securely with other services. Thus, we must be able to (1) name the service with an assertable identity and (2) grant rights to that identity.
3. *Dynamic establishment of trust domains*. In order to coordinate resources, VOs need to establish trust not only between users in the VO and the resources, but also among the resources. These trust domains can span multiple organizations and must adapt dynamically as participants join, are created, or leave the VO.

Traditional means of security administration involving manual editing of policy databases or issuance of credentials cannot meet the demands of these dynamic scenarios. We require a user-driven security model that allows users to create entities and policy domains in order to create and coordinate resources within VOs.

3 GT2 Grid Security Model

To set the stage for our discussion of OGSA and GT3 security, we review briefly the security technologies incorporated in the Globus Toolkit version 2 (GT2) [13] and explain how these address the three key issues introduced above. GT2 includes services for Grid Resource Allocation and Management (GRAM), Monitoring and Discovery (MDS), and data movement (GridFTP). These services use a common Grid Security Infrastructure (GSI) [4, 15] to provide security functionality.

Diverse site security mechanisms. GSI defines a common credential format based on X.509 identity certificates [5, 28] and a common protocol based on transport layer security (TLS[10], SSL [18]). An X.509 certificate, in conjunction with an associated private key, forms a unique credential set that a Grid entity (service or user) uses to authenticate itself to other Grid entities; the TLS-based protocol is used to perform authentication and then provide message protection (encryption, integrity checking) as desired on the subsequent data stream. Gateways are then used to translate between this common GSI infrastructure and local site mechanisms. For example, the Kerberos Certificate Authority (KCA) [29] and SSLK5/PKINIT provide translation from Kerberos to GSI and back, respectively. These mechanisms allow a site with an existing Kerberos installation to continue to use that installation and convert credentials between Kerberos and GSI as needed.

Each GSI certificate is issued by a trusted party known as a certificate authority (CA), usually run by a large organization or commercial company. In order to trust the X.509 certificate presented by an entity, one must trust the CA that issued that certificate. We chose to use X.509 identity certificates within GSI because establishment of this trust is relatively lightweight. In contrast to mechanisms such as Kerberos [25], where trust must be established bilaterally, requiring an agreement at the organizational level, trust in a CA can be established unilaterally. A single entity in an organization can decide to trust any CA, without necessarily involving the organization as a whole. This is a key to the establishment of VOs that involve only some portions of an organization that may receive little or no support from their organization.

Dynamic creation of entities and the granting of privileges to those entities. GSI introduces X.509 proxy certificates, a GSI extension to X.509 identity certificates [28] that allow a user to assign dynamically a new X.509 identity to an entity and then delegate some subset of their rights to that identity. Proxy certificates are created by users issuing a new set of X.509 credentials signed using their own credentials instead of involving a CA. This mechanism allows new credentials and identities to be created quickly without the involvement of a traditional administrator.

Dynamically creation and management of overlaid trust domains. The requirement for overlaid trust domains to establish VOs is satisfied by GSI using both proxy certificates and security services such as the Community Authorization Service (CAS) [26]. Proxy certificates allow for the dynamic creation of trust domains by a user, since they have an implicit policy that two entities bearing proxy certificates issued by the same user will inherently trust each other. Hence, users can issue proxy certificates to any services that they wish so that they can work together and form a trust domain based on the implicit trust policy.

Security services such as CAS allow for flexible, expressive policy to be created regarding multiple users in a VO. CAS allows a VO to express the policy that has been outsourced to it by the resource providers in the VO. As illustrated in Figure 2,

1. The user authenticates to CAS and receive assertions from CAS expressing the VO's policy in terms of how that user may use VO resources.
2. The user then presents the assertion to a VO resource along with the usage request.
3. In evaluating whether to allow the request, the resource checks both local policy and the VO policy expressed in the CAS assertion.

CAS thus allows the resource to remain the ultimate authority over their resource, but it also allows the VO to control some portion of the enforced policy. In turn, the VO can coordinate the policy across a number of resources to control the sharing of those resources by the VO.

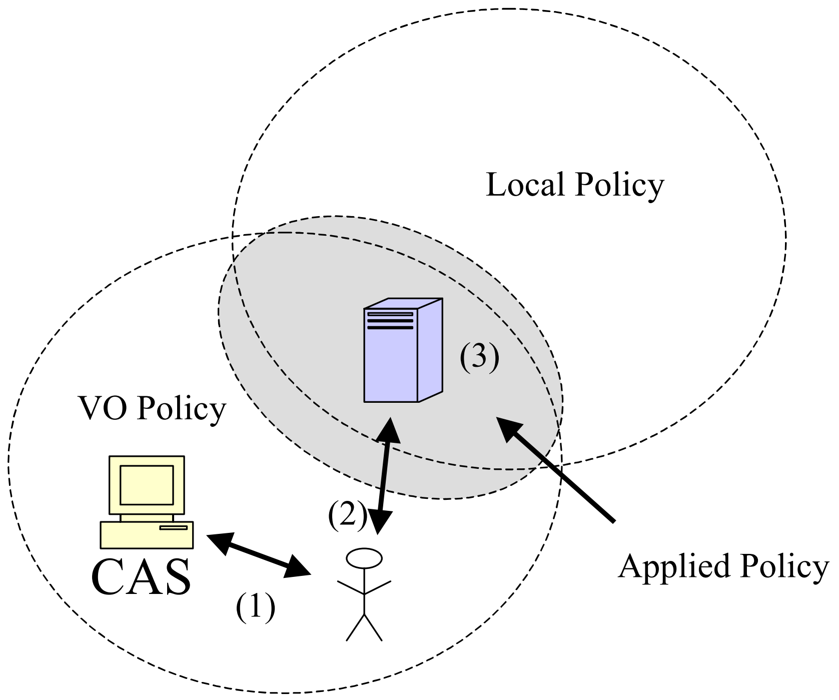

Figure 2: CAS allows VOs to express policy and resources to apply policy that is subset of VO and local policy.

In designing the GSI we evaluated several related efforts before selecting PKI as the basis on which to build for GSI. In much of the existing work, we noted the following short-comings in meeting the Grid security requirements:

- Kerberos [25] requires the explicit involvement of site administrators to establish interdomain trust relationships or to create new entities.
- The CRISIS wide area security system [3] defines a uniform and scalable security infrastructure for wide area systems but does not address interoperability with local security mechanisms.
- Secure SHell (SSH) [30] provides a strong system of authentication and message protection but has no support for translation between different mechanisms and creation of dynamic entities.
- The Legion security model [19] is perhaps the most similar to that of GT2, using, for example, X.509 certificates for delegation- however, it lacks mechanisms for interacting with local site mechanisms.

4 An OGSA Security Model

We now turn to the problem of addressing Grid security challenges within the context of the Open Grid Services Architecture (OGSA) [14], a set of technical specifications that aligns Grid technologies with emerging Web services technologies [18].

"Web services" is used to describe software components in terms of methods for their access, bindings of these methods to specific mechanisms, and methods of discovery of relevant services. In particular, the Simple Object Access Protocol (SOAP) [1] provides a means of messaging using XML envelopes to encapsulate payloads, with HTTP the most commonly used underlying protocol. Another example is the Web Services Description Language (WSDL) [7], which provides a method to express the operation signatures as well as their bindings to protocols and endpoints in an XML document (groups of operations bundled together to form a Web Service).

OGSA defines standard Web service interfaces and behaviors that add to Web Services the concepts of stateful services and secure (when required) invocation (as well as other abilities to address Grid specific requirements, which are not relevant for this paper). These functions define what is called a "Grid Service" and allow users to create and manipulate Grid Services, as allowed by policy, to create sophisticated distributed services. Grid services can define, as part of their interface, service data elements (SDEs) that other entities can (again, subject to policy) query or subscribe to.

OGSA introduces both new opportunities and new challenges for Grid security. Emerging Web services security specifications address the expression of Web service security policy (WS-Policy [2]), standard formats for security token exchange (WS-Security [22]), standard methods for authentication and establishment of security contexts (WS-SecureConversation [20]), and standard methods for the establishment of trust relationships (WS-Trust [21]). These specifications can be exploited, but may in some cases also need to be extended, to address the Grid security requirements listed above.

Version 3 of the Globus Toolkit (GT3) and its accompanying Grid Security Infrastructure (GSI3) provide the first implementation of OGSA mechanisms. GT3's security model seeks to allow applications and users to operate on the Grid in as seamless and automated a manner as possible. Security mechanisms should not have to be instantiated in an application but instead should be supplied by the surrounding Grid infrastructure, allowing it to adapt on behalf of the application to meet its requirements. The application should need to deal with only application-specific policy. GT3 uses the following powerful features of OGSA and Web Services security to work toward that goal:

1. Casting security functionality as OGSA services to allow them to be located and utilized as needed by applications.
2. Use of sophisticated *hosting environments* to handle security for applications and allow security to adapt without having to change the application.
3. Publishing service security policy so that clients can discover dynamically what credentials and mechanisms are needed to establish trust with the service.
4. Specified standards for the exchange of security tokens to allow for interoperability.

In this section we describe how each of these features is used in our OGSA security model and then explain how they are used together to support seamless Grid security.

4.1 Security as Services

Secure operation in a Grid environment requires that application and services have a variety of security functionality, such as authentication, authorization, credential conversion, auditing, and delegation. Grid applications need to interact with other applications and services that have a range of security mechanisms and requirements. These mechanisms and requirements are also likely to evolve over time as new mechanisms are developed or policies change. Grid applications need to avoid having security mechanism statically implemented and must be able to adapt to these changing requirements.

Our OGSA security model casts security functions as OGSA services. This strategy provides well-defined protocols and interfaces for these services and allows an application to outsource security functionality by using a security service with a particular implementation to fit its current need.

The OGSA Security Roadmap [31] itemizes numerous security services. A few examples are listed here:

- *Credential processing:* A service that handles the details of processing and validating authentication tokens
- *Authorization:* A service that takes as input information about a client (identity, attributes, etc.), a service's policy, and details of the clients request and renders a policy decision on the request.
- *Credential Conversion:* A service that allows bridging of different trust or mechanism domains by converting credentials between trust roots or mechanisms.
- *Identity Mapping:* A service that takes a user's identity in one domain and returns the identity in another (e.g. given the user's X.509 identity, it could return the Kerberos principal name).
- *Audit:* A service that accepts information about events and securely logs them appropriately.

4.2 Hosting Environment

Finding and utilizing security services as described in the preceding section require sophistication on the part of the application. Ideally, application developers should not be burdened with the details of this process.

Grid services, like the Web services they leverage, may also be built on sophisticated container-based *hosting environments* such as J2EE or .Net. These hosting environments provide a high level of functionality and allow for the security implementations to be pulled from the applications and placed in the hosting environment. We envision that much of the functionality of security will be placed into the hosting environment, freeing the application of having this implementation and allowing it to be upgraded independently of the application.

4.3 Publishing of Security Policy

In order to establish trust, two entities need to be able to find a common set of security mechanisms that both understand. The use of hosting environments and security services, as described previously in this section, enables OGSA applications and services to dynamically adapt and use different security mechanisms. However, in order to properly select the security mechanisms and credentials to use, an application must know what mechanisms and credentials are acceptable to the service with which it wishes to interact.

The WS-Policy [2] specification and its related specifications define how a Web service can publish its security policy along with its interface specification as part of a WSDL document. Such a published policy can express requirements for mechanisms, acceptable trust roots, token formats, and other security parameters. An application wishing to interact with the service can examine this published policy and gather the needed credentials and functionality by contacting appropriate OGSA security services.

4.4 Specified Format for Security Tokens

The WS-Security [22],WS-SecureConversation [20], and WS-Trust[21] specifications contain formats for the communication of various formats of mechanism-specific tokens (e.g., Kerberos tickets, X.509 certificates) inside SOAP envelopes. This enveloping standardizes the protocol for security mechanisms and allows mechanisms to be independent from any application protocol. Hosting environments can recognize security-related messages and route them to an appropriate service for handling, and entities in the network can recognize if and how an interaction is secured. For example, a firewall can recognize whether a connection is authenticated and then allow only authenticated connections.

4.5 The OGSA Security Model in Action

Figure 3 shows a simplified example of the OGSA security model in action. A request is made by the OGSA client at left to the OGSA service on the right. Both client and service are contained in an advanced hosting environment (Section 4.2), which handle all the security functionality for their contained application and service. For clarity many details of the security process, such as auditing, client-side authorization, and privacy are omitted, but they would function similarly with the hosting environments using OGSA services to provide the needed functionality.

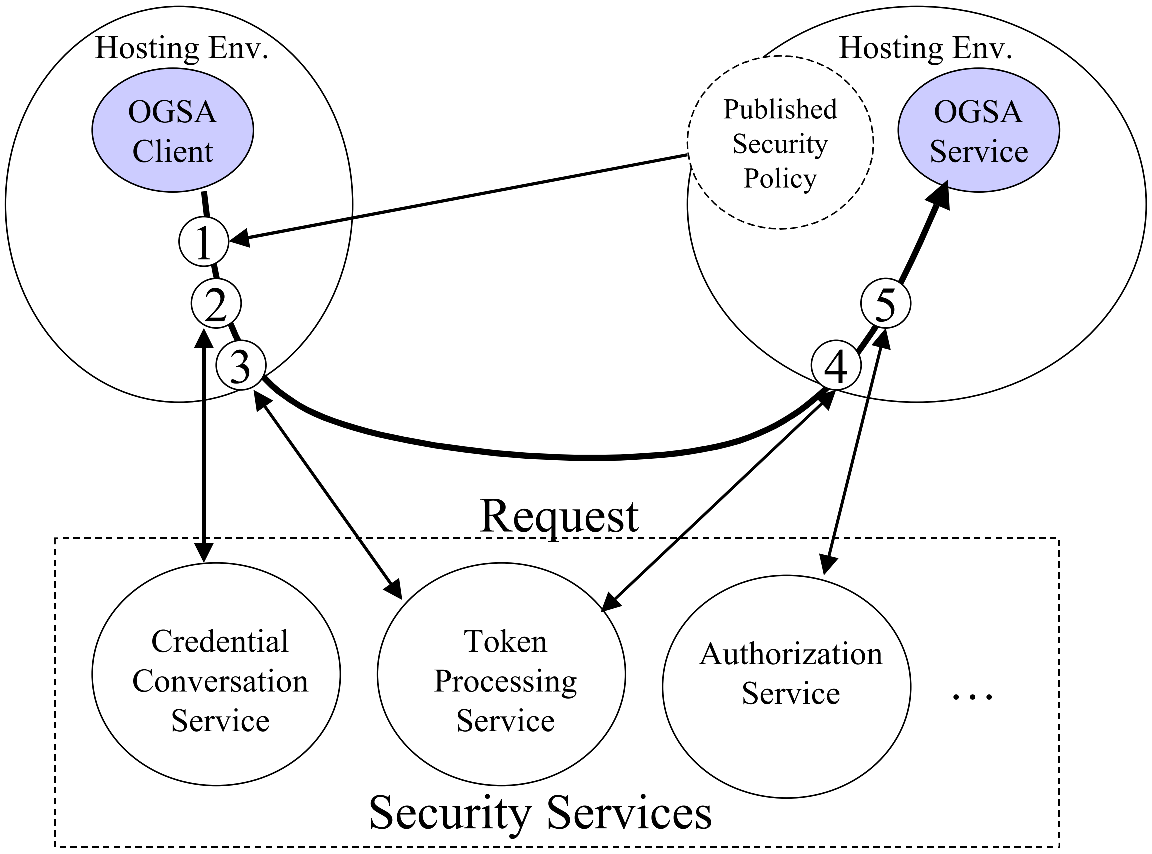

Figure 3: Example of a secured request in the OGSA security model. Steps are described in the text.

The client first forms a request intended for the OGSA service and passes the request to its hosting environment for processing and delivery. The following steps are then taken to handle the security of the request:

1. The client's hosting environment retrieves and inspects the security policy of the target service to determine what mechanisms and credentials are required to submit a request.
2. If the client's hosting environment determines the needed credentials are not already present, it contacts a credential conversion service to convert existing credentials to those of the needed format, mechanism, and/or trust root. Two examples of such services are CAS [26], for translating the user's personal credential to a VO credential, and KCA, for converting between Kerberos and PKI mechanisms.
3. The client's hosting environment uses a token processing and validation service (e.g., XKMS [11]) to handle the formatting and processing of authentication tokens for exchange with the service. This service relieves the application and its hosting environment of having to understand the details of any particular mechanism.

4. On the server side, the hosting environment likewise uses a token processing service to process the authentication tokens presented by the client. (Note that in the example, both use the same service, but each could use a separate service.)
5. After authentication and the determination of client identity and attributes, the service's hosting environment presents the details of the request and client information to an authorization service (e.g., PERMIS [6], Akenti [27]) for a policy decision.

If all the above steps complete successfully, the service's hosting environment then presents the authorized request to the service application. The application, knowing that the hosting environment has already taken care of security, can focus on application-specific request processing steps.

5 GT3 Security Implementation

The Globus Toolkit version 3 (GT3)'s Grid Security Infrastructure version 3 (GSI3) implements key components of the OGSA security model described in Section 4. The resulting system has two key advantages over its GT2 predecessor described in Section 3:

- *Use of WS-Security protocols and standards.* GT3 uses SOAP and the Web Services security specifications for all of its communications. This allows it to leverage and use standard current and future Web Service tools and software.
- *Tight least-privilege model.* In contrast to GT2, the GT3 resource management implementation makes us of *absolutely no* privileged services. All privileged code is contained in two small, tightly constrained setuid programs.

5.1 Use of Web Services Security and Protocol

GT2 uses the TLS transport protocol for both security context establishment and message protection. Establishment of a security context between parties serves both to achieve mutual authentication between the parties and to establish state that is used subsequently for message protection via encryption (for confidentiality) and/or message digests (to prevent tampering).

GT3 uses Web services specifications to allow security messages and secured messaged to be transported, understood and manipulated by standard Web Services tools and software. It achieves security context establishment by implementing WS-SecureConversation [20], which uses SOAP messages to transport context-establishment tokens between the two parties. Our implementation carries the same tokens as in GT2 but transports them over SOAP instead of TCP. Once the context is established, GSI3 implements message protection using the XML-Signature and XML-Encryption specifications.

In addition to the context-based communication described in the preceding section, GT3 offers the ability to secure messages independent of any established security context. Thus, a message can be created and secured without synchronous communication having to be established with the recipient. In the case of a message that is simply to be signed, to allow verification of the message's origin, the identity of the recipient does not have to be known. As we discuss below, this feature allows for messages to be created by clients and delivered to services whose creation is caused by the message itself.

5.2 Least-Privilege Model

The Grid Resource Acquisition and Management (GRAM) [8, 9] is a fundamental GT service. It enables remote clients to securely instantiate, manage and monitor processes on remote resources. Because GRAM enables the instantiation of processes, it is a focal point of the GT security model.

We describe here our GT3 GRAM implementation, focusing on the aspects of the design demonstrating the application of the least privilege model. The key point of our design is the small amount of code that needs to run with privileges and the tight constraints on that code.

5.2.1 GT3 GRAM

GRAM allows remote clients to initiate and manage computational tasks ("jobs") securely on a resource. A client creates a description of the job they wish to have run, specifying such things as the name of the executable, the working directory, where input and output should be stored, and the queue in which it should run. This description is sent to the resource and ultimately results in the creation of an instance of a Managed Job Service (MJS). The MJS is a Grid service that acts as an interface to the job, instantiating the it and then allowing it to be controlled and monitored with standard Grid and Web service mechanisms.

An MJS is created by invoking a create operation on a MJS factory service. While conceptually we want to run one MJS factory service per user account, this is not ideal in practice because it can involve resource consumption by factories that often sit idle when the user is not using the resource. Thus, we introduce the Master Managed Job Factory Service (MMJFS). One MMJFS runs on each resource, in a non-privileged account, and invokes Local Managed Job Factory Services (LMJFS) for users in their account as needed. A service called a Proxy Router exists to route incoming requests from a user to either that user's LMJFS, if present, or the MMJFS, if a LMJFS is not present for the user making the request.

All MJS and MJS factories are implemented as Grid Services running in a hosting environment. Each account has one hosting environment running in it, with MJS instances and a MJS Factory running in that hosting environment. This approach allows for the creation multiple services in a lightweight manner.

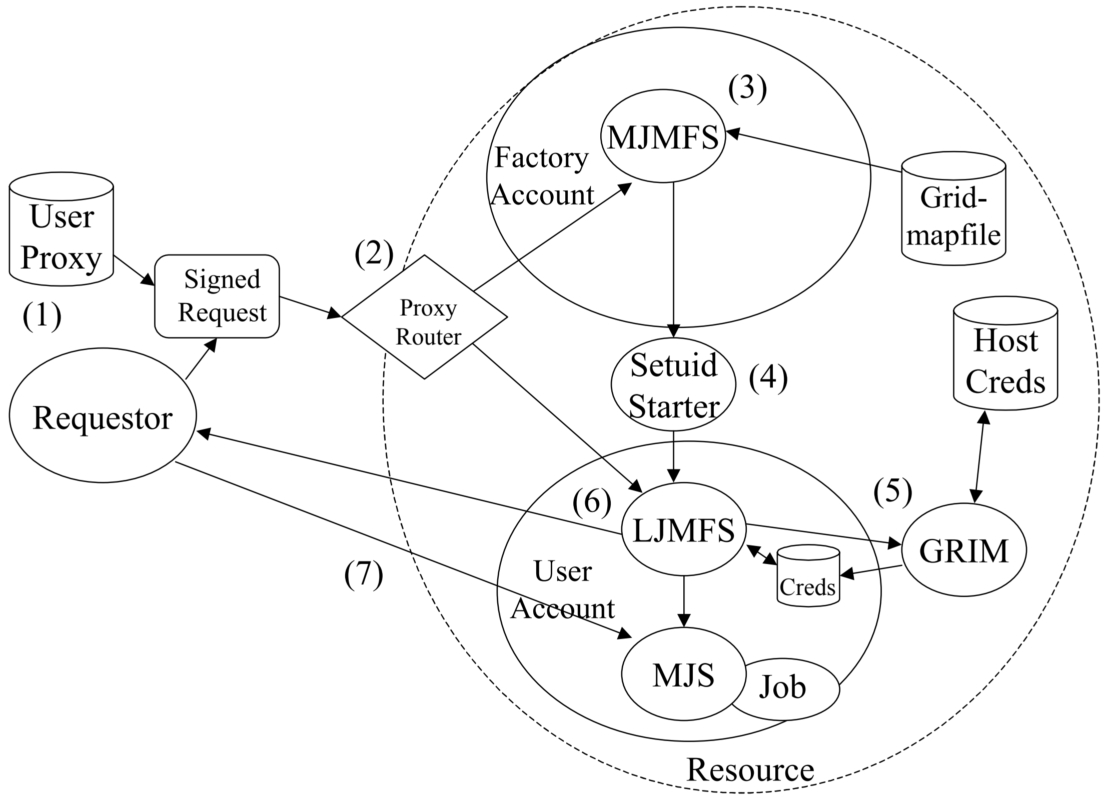

Figure 4: A requestor initiating a job with the GT3 GRAM system. The steps are described in the text.

Figure 4 shows a requestor initiating a job in the GT3 GRAM architecture. On the left is the requestor, with a set of GSI credentials. The resource, with its GRAM services and host credentials, is on the right. Job initiation proceeds as follows.

1. The requestor forms a job description and signs it with their GSI credentials. This request is sent to the target resource on which process initiation is desired.
2. The Proxy Router service accepts the request and either routes it to an LMJFS, if present (skip to step 6), or to the MMJFS otherwise (on to step 3).
3. The MMJFS verifies the signature on the request and establishes the identity of the requestor. It then determines the local account in which the job should be run based on the requestor's identity using the *grid-mapfile*, a local configuration file containing mappings from GSI identities to local identities [4].
4. The MMJFS invokes the *Setuid Starter* process to start a LMJFS for the requestor. The *Setuid Starter* is a privileged program (typically setuid-root) that has the sole function of starting a pre-configued LMJFS for a user.
5. When a LMJFS starts, it needs to acquire credentials and register itself with the Proxy Router. To register, the LMJFS sends a message (not shown) to the Proxy Router. This informs the Proxy Router of the existence of the LMJFS so that it can route future requests for job initiation to it.

The LMJFS invokes the Grid Resource Identity Mapper (*GRIM*) to acquire a set of credentials. *GRIM* is a privileged program (typically setuid-root) that accesses the local host credentials and from them generates a set of GSI proxy credentials for the LMJFS. This proxy credential has embedded in it the user's Grid identity, local account name and local policy to help the client verify that the LMJFS is appropriate for its needs.

6. The LMJFS receives the signed job request. The LMJFS verifies the signature on the request to make sure it has not been tampered with and to verify the requestor is authorized to access the local user account in which the LMJFS is running. Once these verifications are complete, the LMJFS invokes an MJS with the job initiation request and returns the address (GSH) of the MJS to the user.
7. The requestor connects to the MJS to initiate the job. The requestor and MJS perform mutual authentication, the MJS using the credentials acquired from *GRIM*. The MJS verifies the requester is authorized to initiate processes in the local account. The requester authorizes the MJS as having a GRIM credential issued from an appropriate host credential and containing a Grid identity matching its own. This approach allows the client to verify that the MJS it is talking to is running not only on the right host but also in an appropriate account. The user then delegates GSI credentials to the MJS for the job to use and start the job running.

5.2.2 Benefits of GT3 GRAM Model

The GRAM model described in this section has three significant benefits from a least privilege security perspective.

No privileged services. Network services, since they accept and process communications from outside the resource, are more prone to be compromised by logic errors, buffer overflows, and the like. Removing privileges from these services can significantly reduce the impact of compromises by minimizing the privileges gained.

Minimal privileged code. The privileged code is confined to two programs, GRIM and Setuid-Started. The simple and well-constrained functionality of these programs allows them to be audited effectively and reduces the chance they can be used maliciously to gain privilege elevation.

Client-side authorization. GRIM allows the client to verify not only the resource on which an MJS is running on but also the account in which it is running. Thus, a client can act to prevent spoofing of addresses or social engineering tricks that might mislead the user into connecting to, and more importantly delegating credentials to, a MJS other than they intended.

6 Conclusions

Grid computing presents a number of security challenges that are met by the Globus Toolkit's Grid Security Infrastructure (GSI). Version 3 of the Globus Toolkit (GT3) implements the emerging Open Grid Services Architecture and its GSI implementation (GSI3) takes advantage of this evolution to improve on the security model in earlier versions of the toolkit (GT2). GSI3 in GT3 remains compatible (in terms of credential formats) with those used earlier versions, while eliminating privileged network services and making other improvements. Its development provides a basis for a variety of future work. For example, we are interested in exploiting WS-Routing to improve firewall compatibility; in defining and implementing standard services for authorization, credential conversation, identity mapping; and using WS-Policy to allow applications to automatically determine requirements, and locate services to meet those requirements.

Acknowledgements

We are pleased to acknowledge the contributions to the GT3 security design and GSI3 implementation from Doug Engert, Thomas Sandholm, and Rachana Ananthakrishnan.

References

1. Simple Object Access Protocol (SOAP) 1.1, W3C, 2000.
2. BEA, IBM, Microsoft and SAP. Web Services Policy Language (WS-Policy), 2002.
3. Belani, E., Vahdat, A., Anderson, T. and Dahlin, M. The CRISIS Wide Area Security Architecture. *8th Usenix UNIX Security Symposium*, 1998.
4. Butler, R., Engert, D., Foster, I., Kesselman, C., Tuecke, S., Volmer, J. and Welch, V. A National-Scale Authentication Infrastructure. *IEEE Computer*, *33* (12). 60-66. 2000.
5. CCITT Recommendation X.509: The Directory -- Authentication Framework. 1988.
6. Chadwick, D.W. and Otenko, A., The PERMIS X.509 Role Based Privilege Management Infrastructure. *7th ACM Symposium on Access Control Models and Technologies*, 2002.
7. Christensen, E., Curbera, F., Meredith, G. and Weerawarana., S. Web Services Description Language (WSDL) 1.1, W3C, 2001. www.w3.org/TR/wsdl.
8. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S. A Resource Management Architecture for Metacomputing Systems. *4th Workshop on Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, 1998, 62-82.
9. Czajkowski, K., Foster, I., Kesselman, C., Sander, V. and Tuecke, S., SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. *8th Workshop on Job Scheduling Strategies for Parallel Processing*, 2002.
10. Dierks, T. and Allen, C. The TLS Protocol Version 1.0, IETF, 1999. http://www.ietf.org/rfc/rfc2246.txt.
11. Ford, W., Hallam-Baker, P., Fox, B., Dillaway, B., LaMacchia, B., Epstein, J. and Lapp, J. XML Key Management Specification, 2001. www.w3.org/TR/2001.
12. Foster, I. and Kesselman, C. Computational Grids. Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 2-48.
13. Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 259-278.

14. Foster, I., Kesselman, C., Nick, J. and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Globus Project, 2002. www.globus.org/research/papers/ogsa.pdf.
15. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. A Security Architecture for Computational Grids. *ACM Conference on Computers and Security*, 1998, 83-91.
16. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, *15* (3). 200-222. 2001.
17. Gasser, M. and McDermott, E., An Architecture for Practical Delegation in a Distributed System. *Proc. 1990 IEEE Symposium on Research in Security and Privacy*, 1990, IEEE Press, 20-30.
18. Graham, S., Simeonov, S., Boubez, T., Daniels, G., Davis, D., Nakamura, Y. and Neyama, R. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Sams, 2001.
19. Humphrey, M., Knabe, F., Ferrari, A. and Grimshaw, A., Accountability and Control of Process Creation in Metasystems. *2000 Network and Distributed System Security Symposium*, 2000.
20. IBM, Microsoft, RSA Security and VeriSign. Web Services Secure Conversation Language (WS-SecureConversation) Version 1.0, 2002.
21. IBM, Microsoft, RSA Security and VeriSign. Web Services Trust Language (WS-Trust), 2002.
22. IBM, Microsoft and VeriSign. Web Services Security Language (WS-Security), 2002.
23. Linn, J. Generic Security Service Application Program Interface, Version 2. *INTERNET RFC 2078*, 1997.
24. Myers, J. Simple Authentication and Security Layer (SASL), IETF, 1997.
25. Neuman, B.C. and Ts'o, T. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, *32* (9). 33-88. 1994.
26. Pearlman, L., Welch, V., Foster, I., Kesselman, C. and Tuecke, S., A Community Authorization Service for Group Collaboration. *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.
27. Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K. and Essiari, A., Certificate-based Access Control for Widely Distributed Resources. *8th Usenix Security Symposium*, 1999.
28. Tuecke, S., Engert, D., Foster, I., Thompson, M., Pearlman, L. and Kesselman, C. Internet X.509 Public Key Infrastructure Proxy Certificate Profile, IETF, 2001.
29. Kornievskaia, O., Honeyman, P., Doster, B., and Coffman, K., Kerberized Credential Translation: A Solution to Web Access Control. 10th Usenix Security Symposium, 2001.
30. OpenSSH, www.openssh.com, 2003.
31. Siebenlist, F., et al, OGSA Security Roadmap, 2002. http://www.globus.org/ogsa/Security/ogsa-sec-roadmap-v13.pdf